L1: POWER CONSIDERATIONS AND IDE BASICS

ESE516: IoT Edge Computing

Wednesday, January 23, 2019

Eduardo Garcia - edgarc@seas.upenn.edu

AGENDA

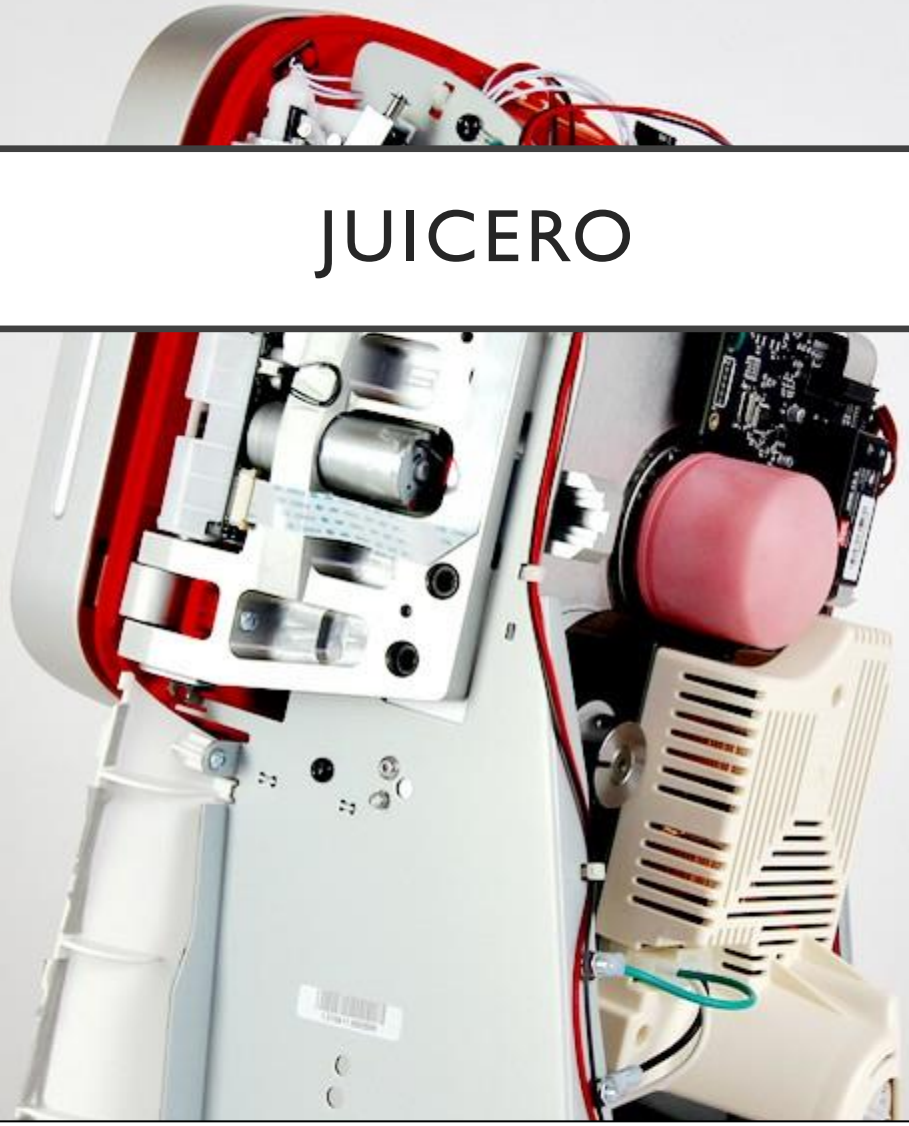
- Bill of Materials (BOM) next steps -> Power
- Tools to develop firmware in industry: Introduction to IDEs and HALs
- Introduction to Atmel Studio 7 and ASF Lab
  - Good embedded coding practice #1 – Comments and Documentation with Doxygen – Moved to Monday 01/28/2019

BOM REVIEW

BOM REVIEW

- Making a tentative BOM for an idea has the following benefits:
  - Answers, "Can a product be made with components on the market?"
    - If not, are we willing to do the components (plus R&D) ourselves?
  - Grounds us to reality
    - Is it feasible to take to market?
    - How much would it cost approximately?
    - Can we get the components needed? What is the market availability

JUICERO



https://blog.bolt.io/heres-why-juicero-s-press-is-so-expensive-6add74594e50

BOM NEXT STEPS - POWER

- Making a tentative BOM allows us to answer a very important question for embedded devices (specially IoT):

How much power do we need?

This question determines:

- Power source sizing
- Can our preferred power source handle this power requirement?
- How long does our product last when on?

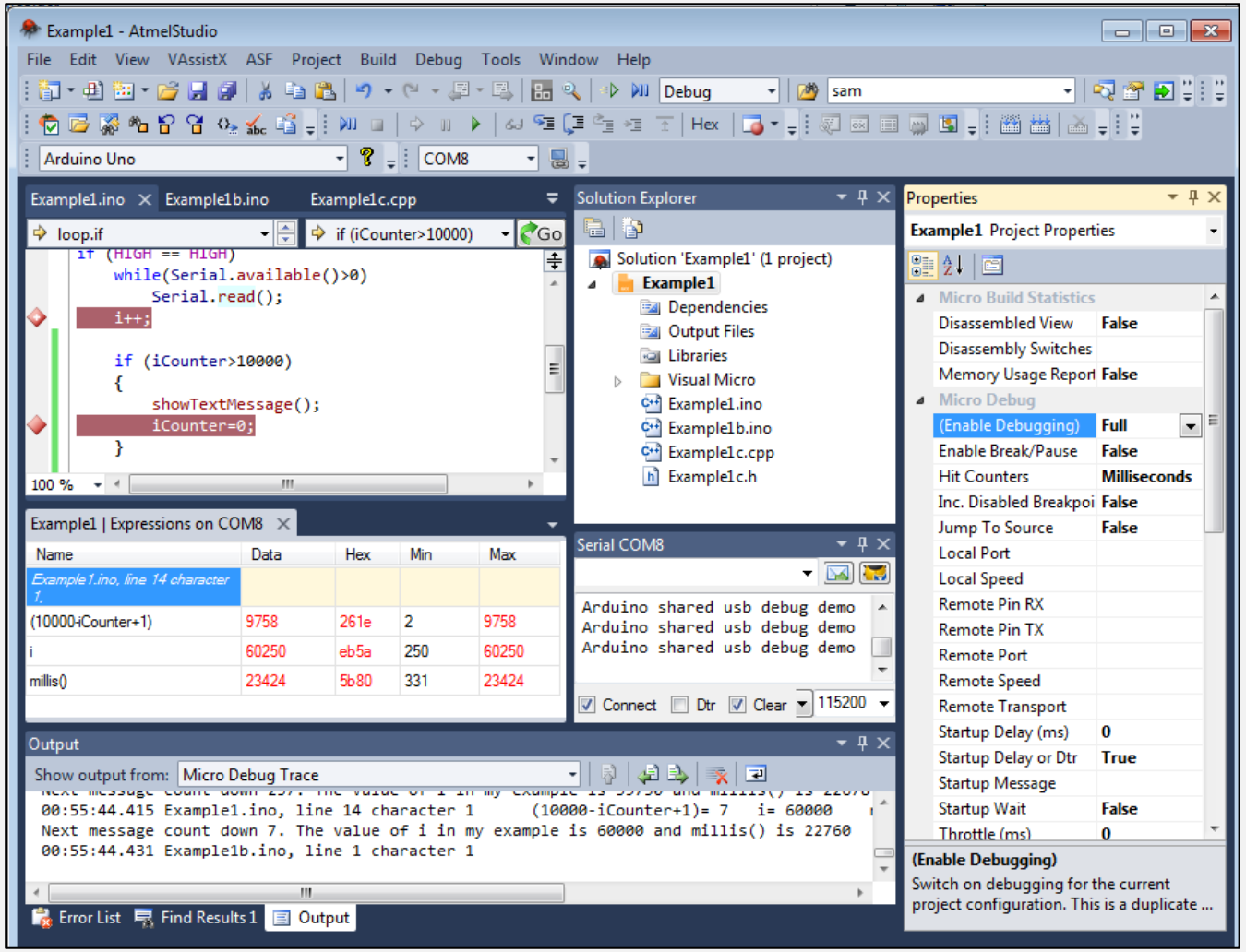
BOM NEXT STEPS - POWER

- For next Monday (January 28th, 2019):
  - Power budget for device
  - Power architecture (Do we need a Buck and/or a boost converter? LDOs?
  - More detail in project deliverable document

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

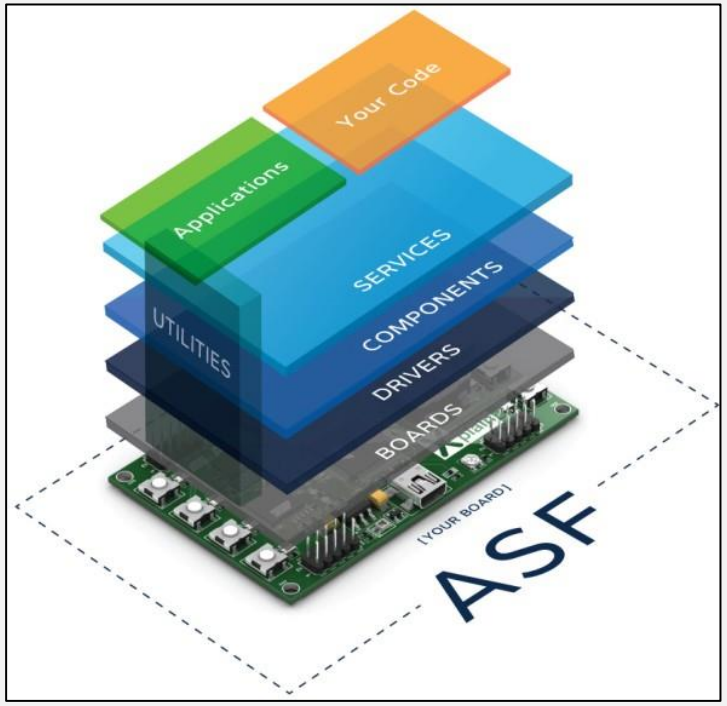
WHAT'S AN IDE?

- Swiss army knife for development
  - Code Editor
  - Build automation tools
  - Debugging
- Household names: Visual Studio, Eclipse, Xcode
- Atmel Studio 7 is our choice
  - Free to use, built for our MCU
  - Only works with Atmel MCUs



WHY ATMEL STUDIO?

- Supports the SAM W25
- Free, unlimited use
- Atmel Studio Framework (ASF) – libraries baked into the IDE to simplify code development
- Similar to Visual Studio
- Comprehensive toolset – disassembler, variable watch, register display, debug breakpoints, etc.



ATMEL STUDIO FRAMEWORK (ASF)

- Stand on the shoulders of giants
- Documentation and examples
- Well vetted code
  (but there still can be errors!)
- Built for ease of use in many scenarios –
  not the slimmest codebase

- **USB**
  - USB Host Interface
  - USB Device Controller
  - HID generic
  - HID keyboard
  - HID mouse
  - PHDC
  - MSD (Mass Storage Device)
  - CDC (Communication Class Driver)
- **Serial Peripherals**
  - SPI
  - I^2C
  - UART
  - CAN
  - LIN
- **TCP/IP (with lwIP examples)**
- **File System (with FatFS examples)**
- **QTouch / Touch Sensors**
- **Cryptography**
- **Power Management**
- **Displays / Graphics**
  - LCD Controllers
  - OLED Controllers
  - Touch Screens
  - Character Displays
  - Various Graphics Libraries
- **Non-volatile Memory**
  - DataFlash
  - SD/MMC Drivers
  - NAND Flash
  - EEPROM
- **Other Peripherals**
  - GPIO
  - ADC
  - DAC
  - Audio
  - DMA Controllers
  - Clock Controllers
  - Watchdog Timers
  - Peripheral Timers
  - PWM
  - Etc.

ASF VS DIRECT REGISTER CODING

- Coding registers directly

```
27 void init_TC3()
28 {
29     /* Configure Timer/Counter 3 as a timer to blink LED0 */
30     // Configure Clocks
31     REG_GCLK_CLKCTRL = GCLK_CLKCTRL_CLKEN | GCLK_CLKCTRL_GEN_GCLK0 | GCLK_CLKCTRL_ID_TCC2_TC3;
32     REG_PM_APBCMASK |= PM_APBCMASK_TC3; // Enable TC3 bus clock
33
34     // Configure TC3 (16 bit counter by default)
35     REG_TC3_CTRLA |= TC_CTRLA_PRESCALER_DIV8;
36
37     // Enable interrupts
38     REG_TC3_INTENSET = TC_INTENSET_OVF | TC_INTENSET_ERR;
39
40     // Enable TC3
41     REG_TC3_CTRLA |= TC_CTRLA_ENABLE;
42     while ( TC3->COUNT16.STATUS.bit.SYNCBUSY == 1 ){} // wait for TC3 to be enabled
43 }
```

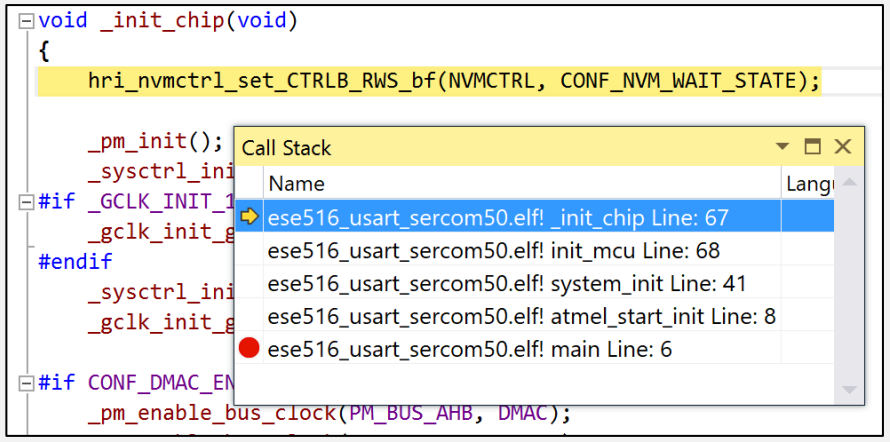
- Coding with a HAL (Atmel ASF)

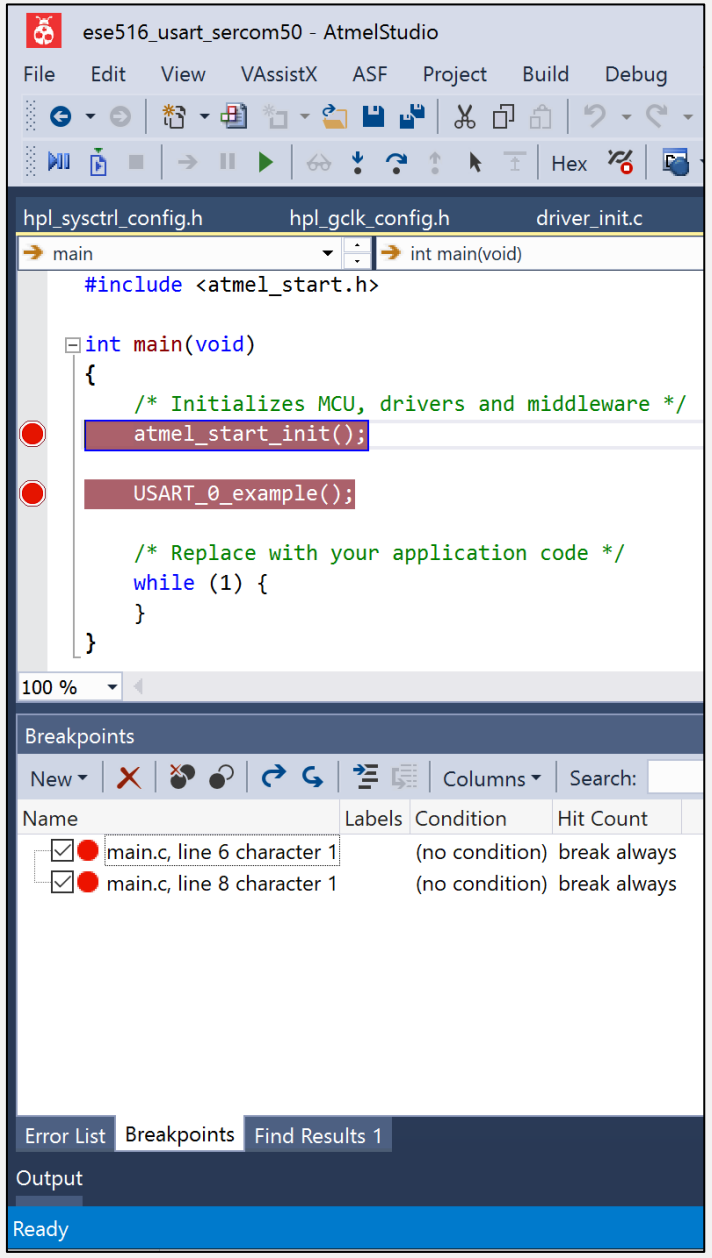
```
struct tcc_module tcc_instance;


static void configure_tcc(void)
{
    struct tcc_config config_tcc;
    tcc_get_config_defaults(&config_tcc, TCC0);
    #if (SAMR30E)
    config_tcc.counter.clock_source = GCLK_GENERATOR_0;
    #else
    config_tcc.counter.clock_source = GCLK_GENERATOR_1;
    #endif
    config_tcc.counter.clock_prescaler = TCC_CLOCK_PRESCALER_DIV64;
    config_tcc.counter.period =   2000;
    config_tcc.compare.match[0] =  900;
    config_tcc.compare.match[1] =  930;
    config_tcc.compare.match[2] = 1100;
    config_tcc.compare.match[3] = 1250;
    tcc_init(&tcc_instance, TCC0, &config_tcc);
    tcc_enable(&tcc_instance);
}
```

SCENARIO: BREAKPOINTS

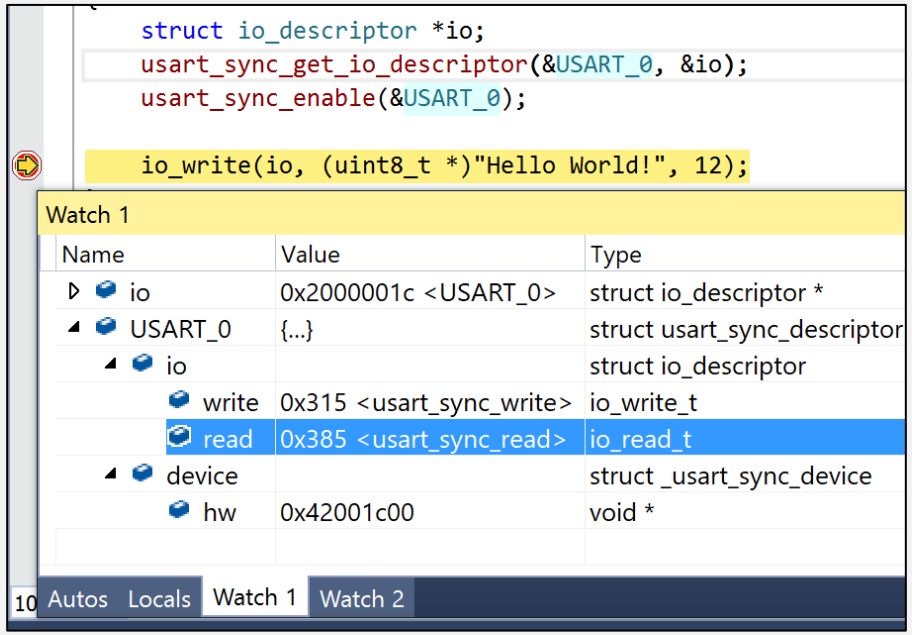
- Your code is running fine until you press a button – then, it seems to hang.
- You set a breakpoint at the button ISR, and follow from there.
- Call stack provides an execution trail





SCENARIO: VARIABLE WATCH

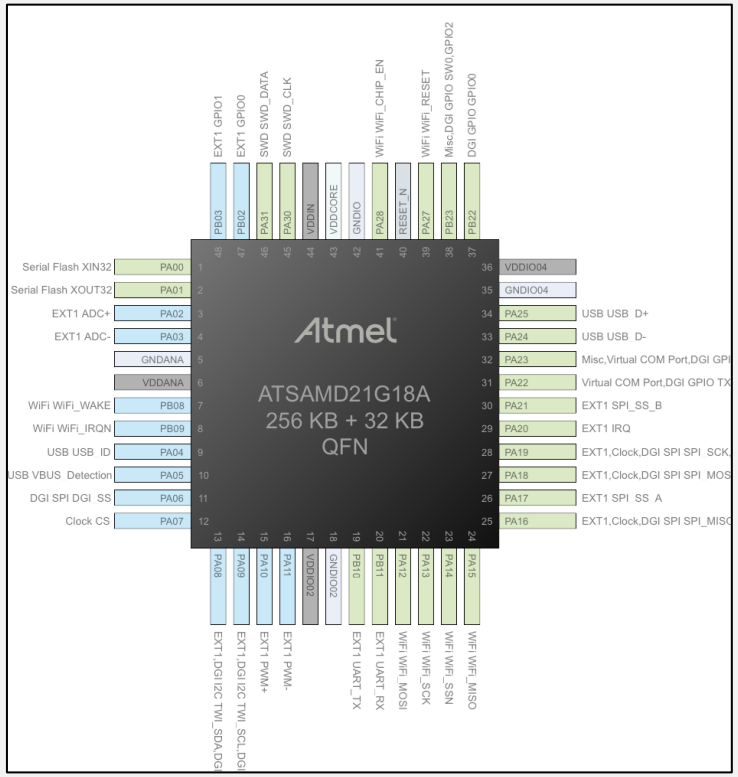
- You've created a function that performs mathematics on two timestamps. However, it keeps giving you the wrong results.
- After setting a breakpoint, you add a number of variables to your watch list. You realize that the timestamps were 32-bit, but the output was stored as 16-bit.



PIN MAPPING

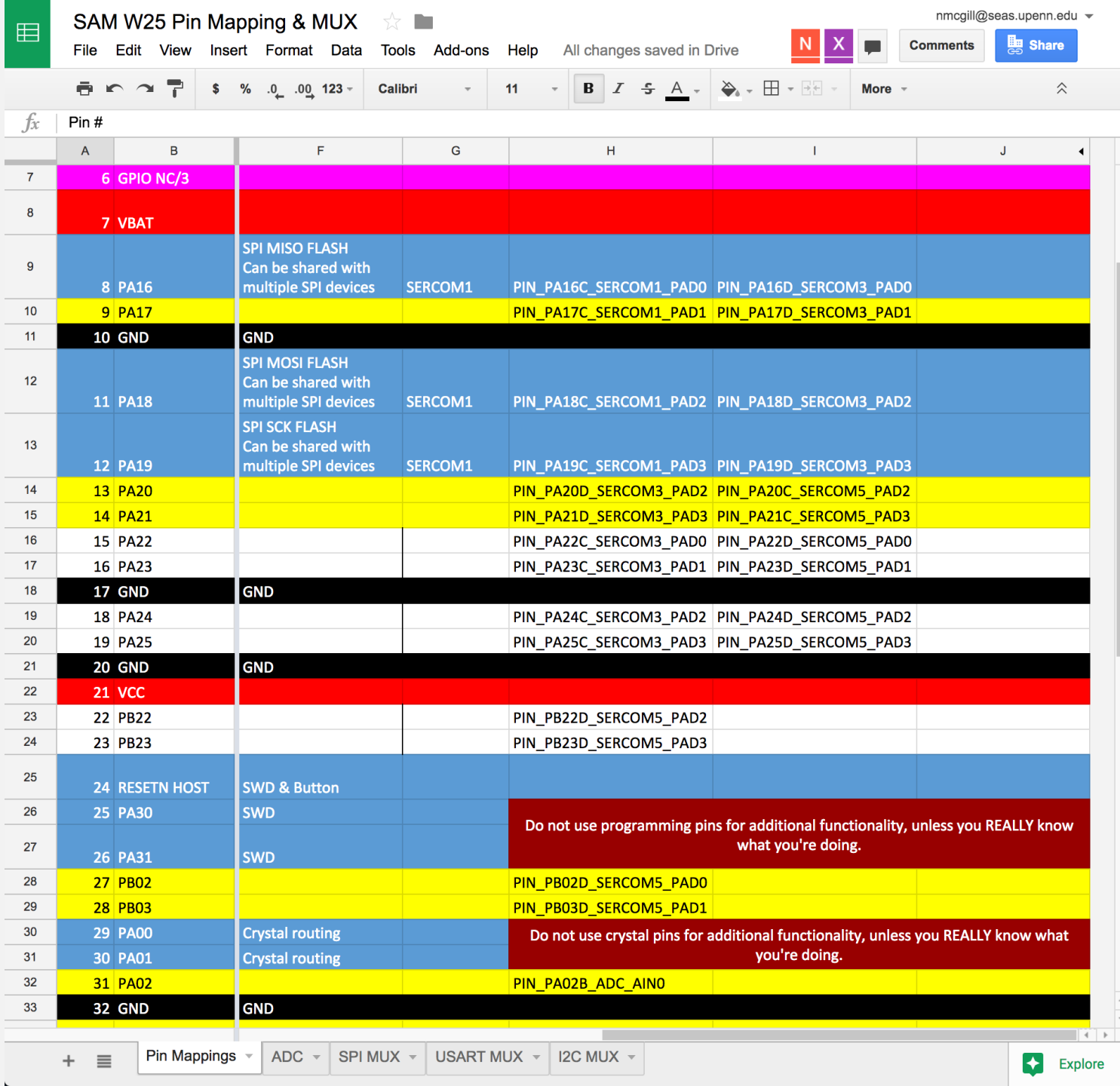
PIN MUXING: SERCOM

- Atmel SAM (ARM) ICs have some flexibility in terms of pin mapping
- SERCOM = **SER**ial **COM**munication
  - Supports SPI, USART, I2C
  - I2S not a part of this
- 6x SERCOM# available for use
  - Cannot reuse SERCOM numbers – for example is SERCOM0 is your USART debug, you cannot also use SERCOM0 as your I2C bus



PIN MUXING

- Sheets document helping track all of these auxiliary functions
- Many pins have two SERCOMs they can operate on
- All pins will retain the same PAD#
- SERCOM# will have a group of pins that work together
- Unsure of your pin mapping?
  - Read up those datasheets
  - Check start.atmel.com
  - **More on this next class!**



	A	B	F	G	H	I	J
7	6	GPIO NC/3					
8	7	VBAT					
9	8	PA16	SPI MISO FLASH Can be shared with multiple SPI devices	SERCOM1	PIN_PA16C_SERCOM1_PAD0	PIN_PA16D_SERCOM3_PAD0	
10	9	PA17			PIN_PA17C_SERCOM1_PAD1	PIN_PA17D_SERCOM3_PAD1	
11	10	GND	GND				
12	11	PA18	SPI MOSI FLASH Can be shared with multiple SPI devices	SERCOM1	PIN_PA18C_SERCOM1_PAD2	PIN_PA18D_SERCOM3_PAD2	
13	12	PA19	SPI SCK FLASH Can be shared with multiple SPI devices	SERCOM1	PIN_PA19C_SERCOM1_PAD3	PIN_PA19D_SERCOM3_PAD3	
14	13	PA20			PIN_PA20D_SERCOM3_PAD2	PIN_PA20C_SERCOM5_PAD2	
15	14	PA21			PIN_PA21D_SERCOM3_PAD3	PIN_PA21C_SERCOM5_PAD3	
16	15	PA22			PIN_PA22C_SERCOM3_PAD0	PIN_PA22D_SERCOM5_PAD0	
17	16	PA23			PIN_PA23C_SERCOM3_PAD1	PIN_PA23D_SERCOM5_PAD1	
18	17	GND	GND				
19	18	PA24			PIN_PA24C_SERCOM3_PAD2	PIN_PA24D_SERCOM5_PAD2	
20	19	PA25			PIN_PA25C_SERCOM3_PAD3	PIN_PA25D_SERCOM5_PAD3	
21	20	GND	GND				
22	21	VCC					
23	22	PB22			PIN_PB22D_SERCOM5_PAD2		
24	23	PB23			PIN_PB23D_SERCOM5_PAD3		
25	24	RESETN HOST	SWD & Button				
26	25	PA30	SWD		Do not use programming pins for additional functionality, unless you REALLY know what you're doing.		
27	26	PA31	SWD				
28	27	PB02			PIN_PB02D_SERCOM5_PAD0		
29	28	PB03			PIN_PB03D_SERCOM5_PAD1		
30	29	PA00	Crystal routing		Do not use crystal pins for additional functionality, unless you REALLY know what you're doing.		
31	30	PA01	Crystal routing				
32	31	PA02			PIN_PA02B_ADC_AIN0		
33	32	GND	GND				

PIN MUXING: SPI

- DOPO = Data Out Pin Out
- DIPO = Data In Pin Out

10.1. Master Mode Settings

The following table describes the SERCOM pin functionalities for the various MUX settings, whilst in SPI Master mode.

Note: If MISO is unlisted, the SPI receiver must not be enabled for the given MUX setting.

Combination	DOPO / DIPO	SERCOM PAD[0]	SERCOM PAD[1]	SERCOM PAD[2]	SERCOM PAD[3]
A	0x0 / 0x0	MOSI	SCK	-	-
B	0x0 / 0x1	MOSI	SCK	-	-
C	0x0 / 0x2	MOSI	SCK	MISO	-
D	0x0 / 0x3	MOSI	SCK	-	MISO
E	0x1 / 0x0	MISO	-	MOSI	SCK
F	0x1 / 0x1	-	MISO	MOSI	SCK
G	0x1 / 0x2	-	-	MOSI	SCK
H	0x1 / 0x3	-	-	MOSI	SCK
I	0x2 / 0x0	MISO	SCK	-	MOSI
J	0x2 / 0x1	-	SCK	-	MOSI
K	0x2 / 0x2	-	SCK	MISO	MOSI
L	0x2 / 0x3	-	SCK	-	MOSI
M	0x3 / 0x0	MOSI	-	-	SCK
N	0x3 / 0x1	MOSI	MISO	-	SCK
O	0x3 / 0x2	MOSI	-	MISO	SCK
P	0x3 / 0x3	MOSI	-	-	SCK

PIN MUXING: USART

- If RX & TX are connected, operation is in half duplex
- XCK can be used for clock sync
  - Check special case with RX & XCK sharing pin
  - For our CLI, we're going to safely ignore this.
  - You may have a specific reason for it!
- Your CLI UART will be full duplex.

10. SERCOM USART MUX Settings

The following lists the possible internal SERCOM module pad function assignments, for the four SERCOM pads when in USART mode. Note that this is in addition to the physical GPIO pin MUX of the device, and can be used in conjunction to optimize the serial data pin-out.

When TX and RX are connected to the same pin, the USART will operate in half-duplex mode if both one transmitter and several receivers are enabled.

Note: When RX and XCK are connected to the same pin, the receiver must not be enabled if the USART is configured to use an external clock.

MUX/Pad	PAD 0	PAD 1	PAD 2	PAD 3
RX_0_TX_0_XCK_1	TX / RX	XCK	-	-
RX_0_TX_2_XCK_3	RX	-	TX	XCK
RX_1_TX_0_XCK_1	TX	RX / XCK	-	-
RX_1_TX_2_XCK_3	-	RX	TX	XCK
RX_2_TX_0_XCK_1	TX	XCK	RX	-
RX_2_TX_2_XCK_3	-	-	TX / RX	XCK
RX_3_TX_0_XCK_1	TX	XCK	-	RX
RX_3_TX_2_XCK_3	-	-	TX	RX / XCK

PIN MUXING: I2C

- Only one configuration for I2C muxing
  - PAD0 = SDA
  - PAD1 = SCL
- Other two pads should be declared unused for the SERCOM.
  - Can be used as a GPIO, ADC, or other non-SERCOM function

uint32_t	pinmux_pad0	PAD0 (SDA) pinmux
uint32_t	pinmux_pad1	PAD1 (SCL) pinmux

L2.0: ATMEL STUDIO & ASF

L2.0: ATMEL STUDIO & ASF

- Go through the rest of these slides – read through everything.
- Open Atmel Studio and poke around. Get comfortable.
- Don't be afraid to read code. Get to know it.
  - Use "Find References" on functions to determine their function and routes.
- **ASF Explorer** has APIs and Quick Start Guides for all modules

Comprehension Questions:

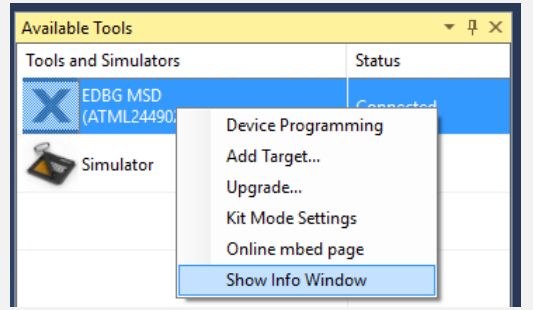
- Do you understand how to set up a project? Which MCU do you select?
- What is an Atmel ICE? Why don't we need it for your SAM W25 Xplained boards?

GETTING STARTED – EXPLORING THE IDE

GETTING STARTED

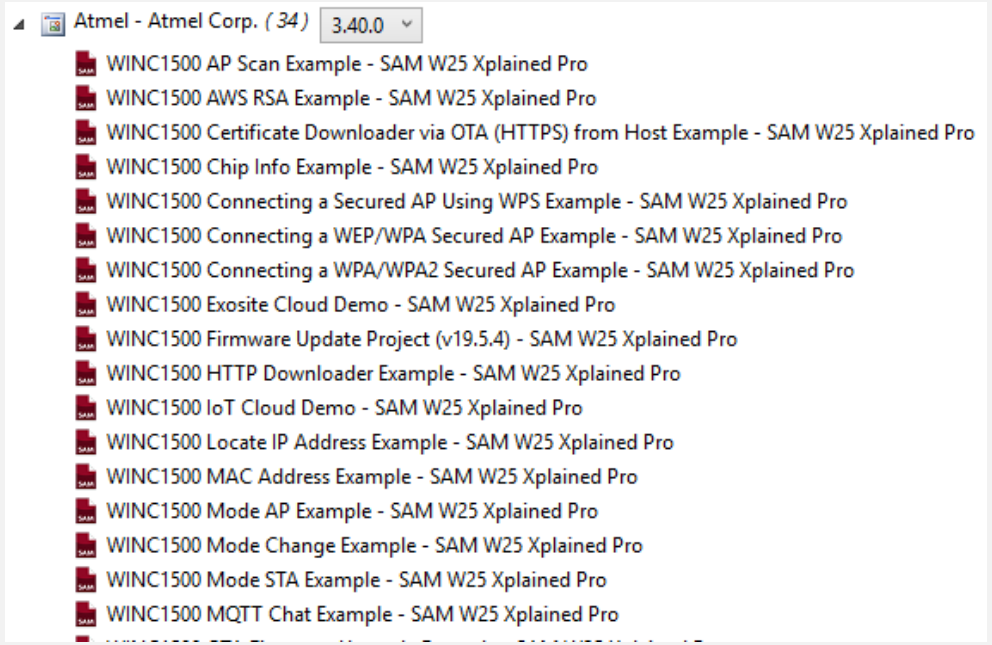
- Connect the SAMW25 board to the computed using the USB cable (use the port on the board that says "DEBUG USB")
- Open Atmel Studio
- You will be greeted by the Initial SAMW25 greeting page as shown on the left.

- *If you do not see this, go to "View >> Available Atmel Tools". A panel should appear – right click "EDBG MSD >> Show Info Window"*



GETTING STARTED

- Click on "New ASF Example Project…"
  - *This will load a new window that will show you all the example project related to the WINC1500. As you can see, we have a lot of examples that exercise the Wi-Fi module!*



BEFORE GETTING STARTED…

- However, before we start we must deal with a reality that is too frequent on embedded systems with Wi-Fi modules…

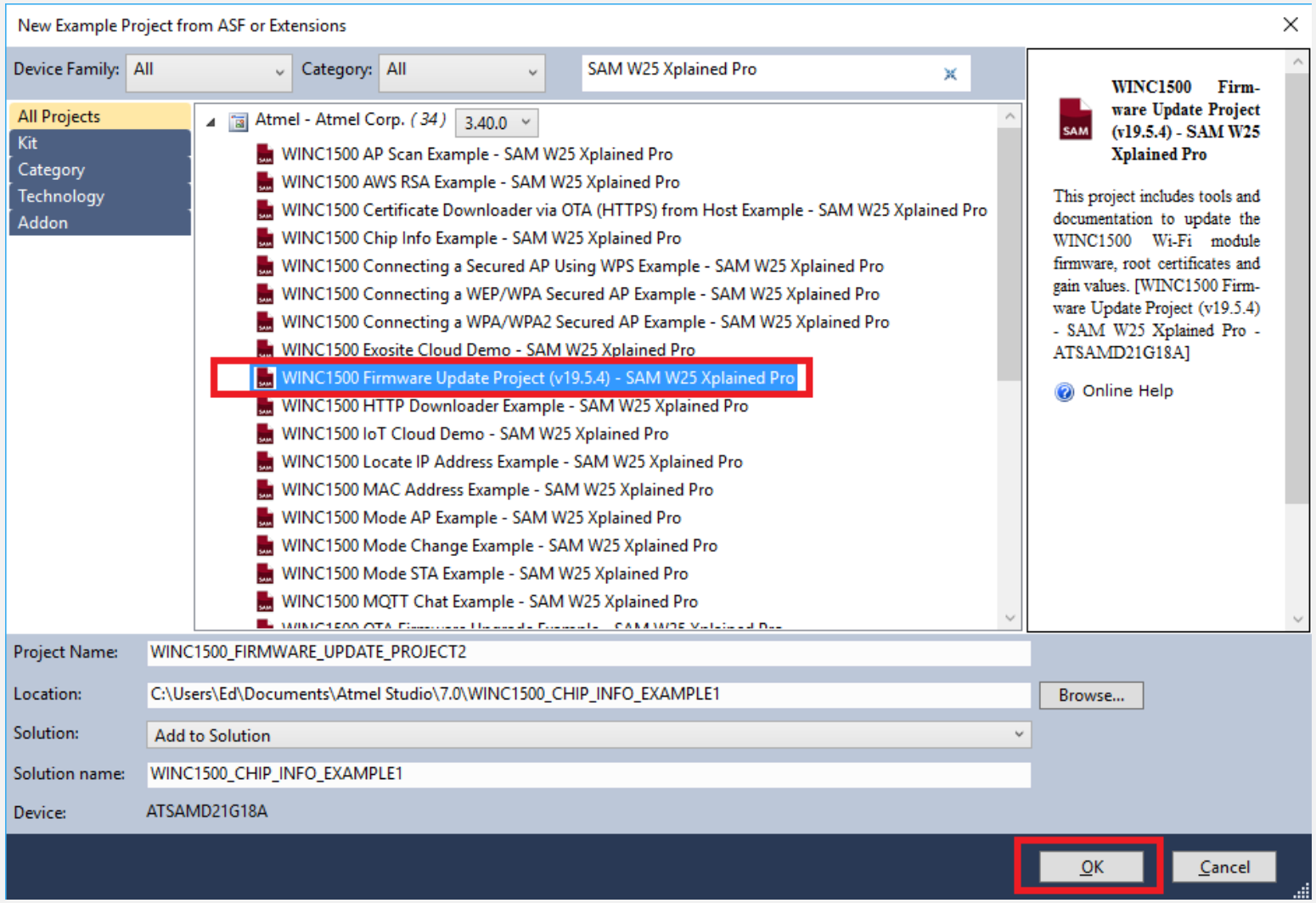
- ***Most probably, the firmware on your WINC1500 is out of date and must be updated! No worries, the next example will show you how to do this.***

- *This will show you the general workflow of loading a project and running it.* ***We only need to do this process once to upgrade your WINC1500!***

BEFORE GETTING STARTED

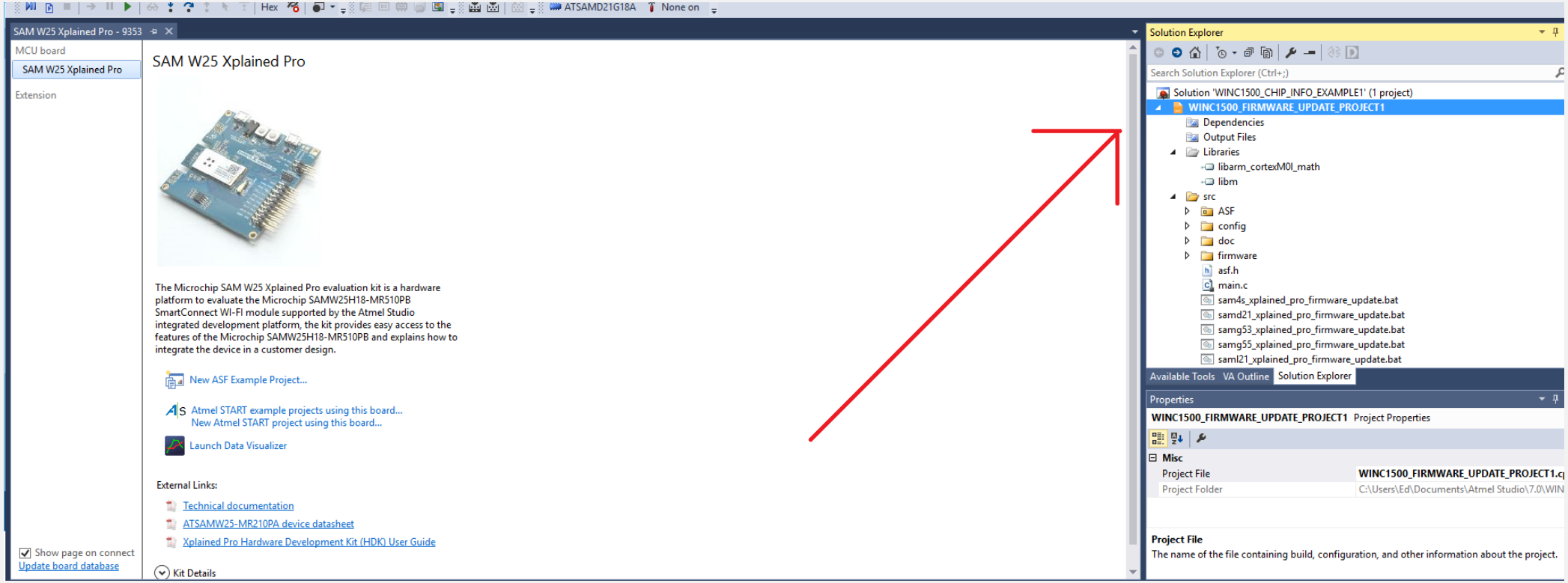
- Open the example "WINC1500 Firmware Update Project (v19.5.4) – SAMW25 Xplained Board Pro" and hit OK.

- *Hit "Accept" to any other windows that appear asking you to accept the terms and services.*



BEFORE GETTING STARTED

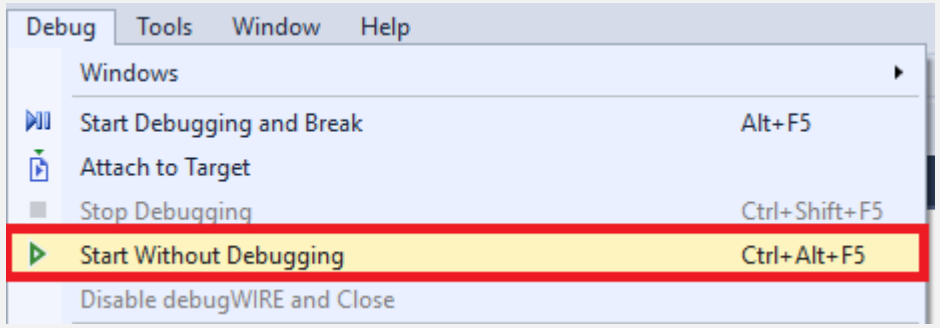
- The project will be loaded on the solution explorer. You can expand the folders to view the files that comprise the project.

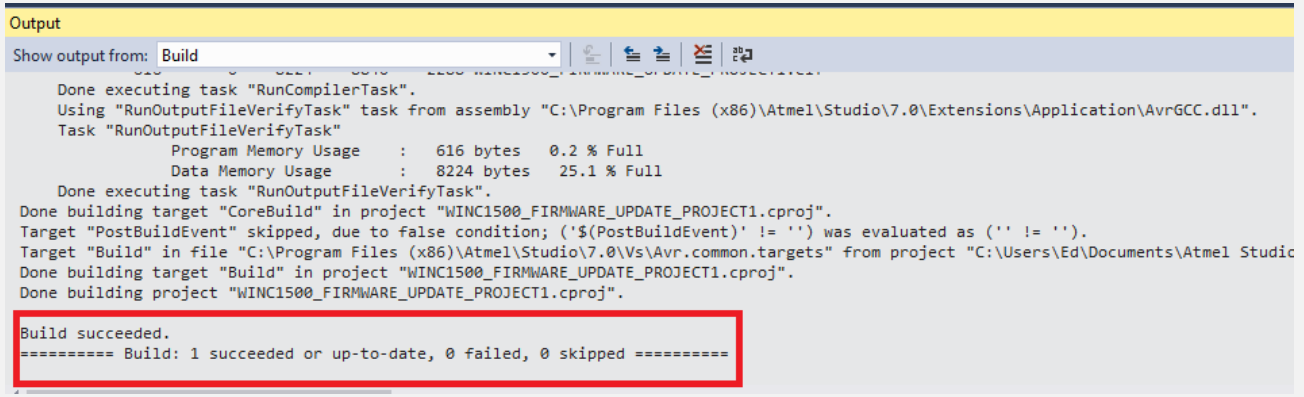


BEFORE GETTING STARTED

- Now that we have a project open, let's load it into the SAMW25 and run it! For now we will only load the code and tell it to run – we will learn how to debug in a later session. This particular code will fail if we run it in debugger mode (the WINC1500 will not be able to update)

- To build the code and load it, go to "Debug >> Start Without Debugging". This will build the code and load it into the MCU. If successful, you will see the "Build succeeded" on the "Output" window.
- *Note: If you get a window asking you to upgrade the debugger, hit "upgrade" and wait until it finishes, then try the previous step again.*





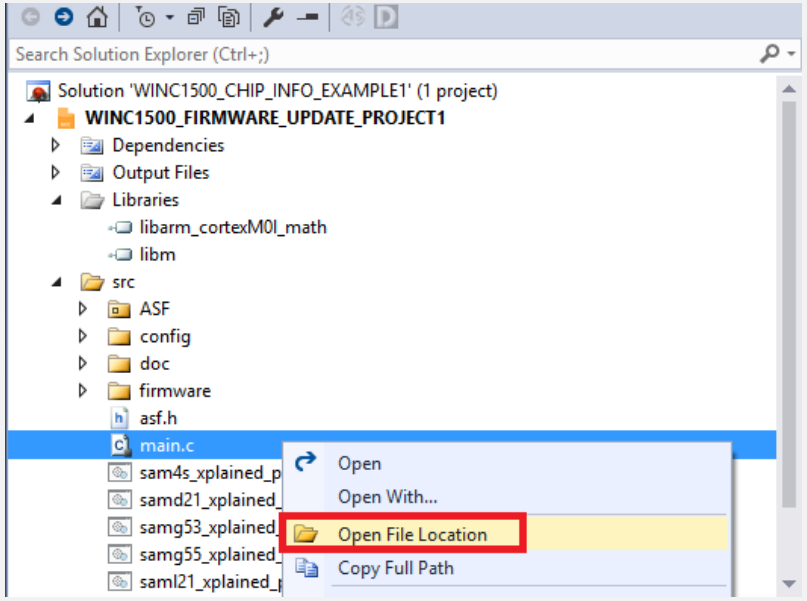
BEFORE GETTING STARTED

- Now you have a program running! The previous method shows you how to open a project and load it into an MCU. Before we continue to more exiting stuff, we need to finish upgrading the WINC1500

- *What we just did was upload a program that will allow us to upgrade the WINC1500. It is a program that turns the SAMD21, our MCU, into a communication bridge between our PC and the WINC1500*

BEFORE GETTING STARTED

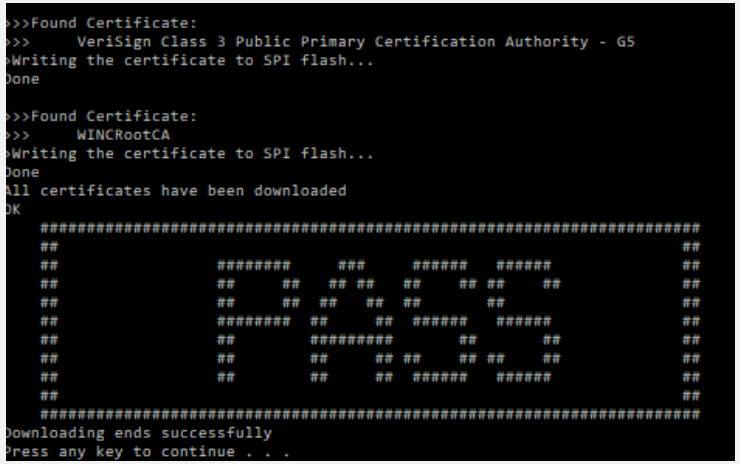
- Now, search for "main.c" on the Solution Explorer and right click it. Select "Open File Location"
  - *This is a good way to find where your project files are!*



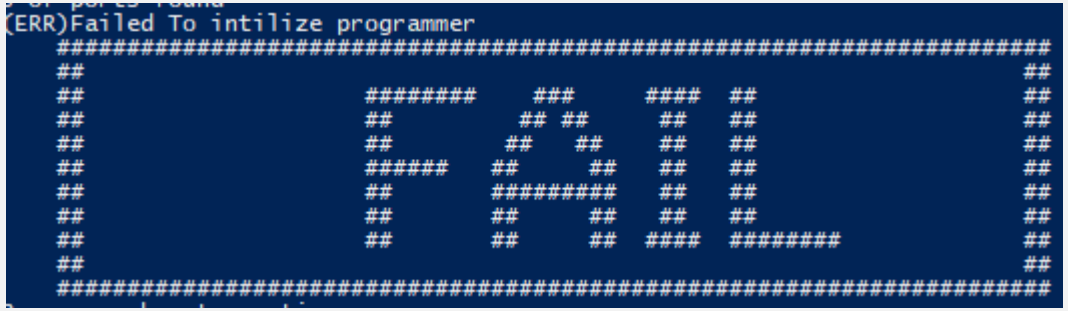
- We need to run the batch file "samw25_xplained_pro_firmware_update.bat" to send the update. Double click this file. A CMD window should appear and take ~20 seconds outputting tests.

BEFORE GETTING STARTED

- IF YOU SEE "PASS" – Continue!



- IF YOU SEE "FAIL" – Call me or call TAs!



WINC1500 UPDATED
HOPEFULLY WE NEVER HAVE TO DO THAT EVER AGAIN

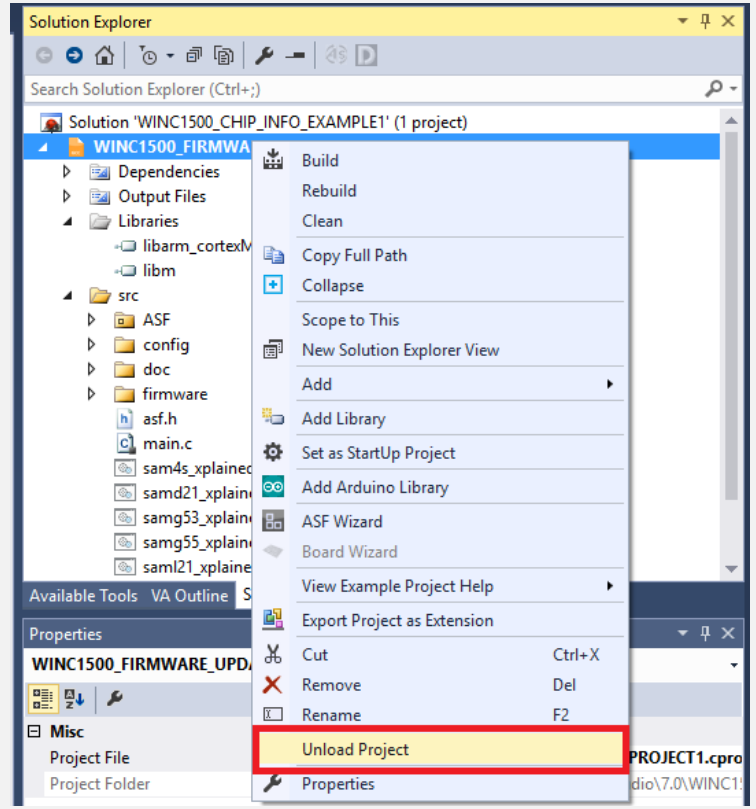
RUNNING AN WINC EXAMPLE – DEBUGGING AND BREAKPOINTS

If you are familiar with IDEs, proceed to next section.

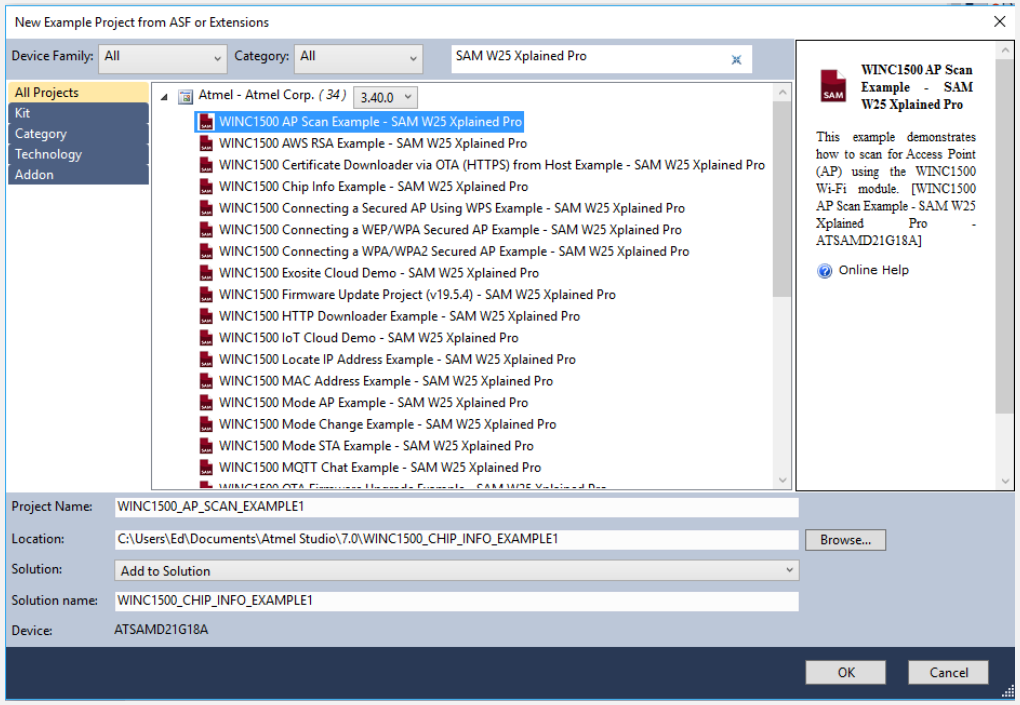
However, the example we are using here might be of use for later assignments...!

RUNNING A WINC1500 EXAMPLE

- Unload the "Firmware Update" project (right click on Project and do "Unload Project".
  - You can also just remove it by doing "Right Click >> Remove"

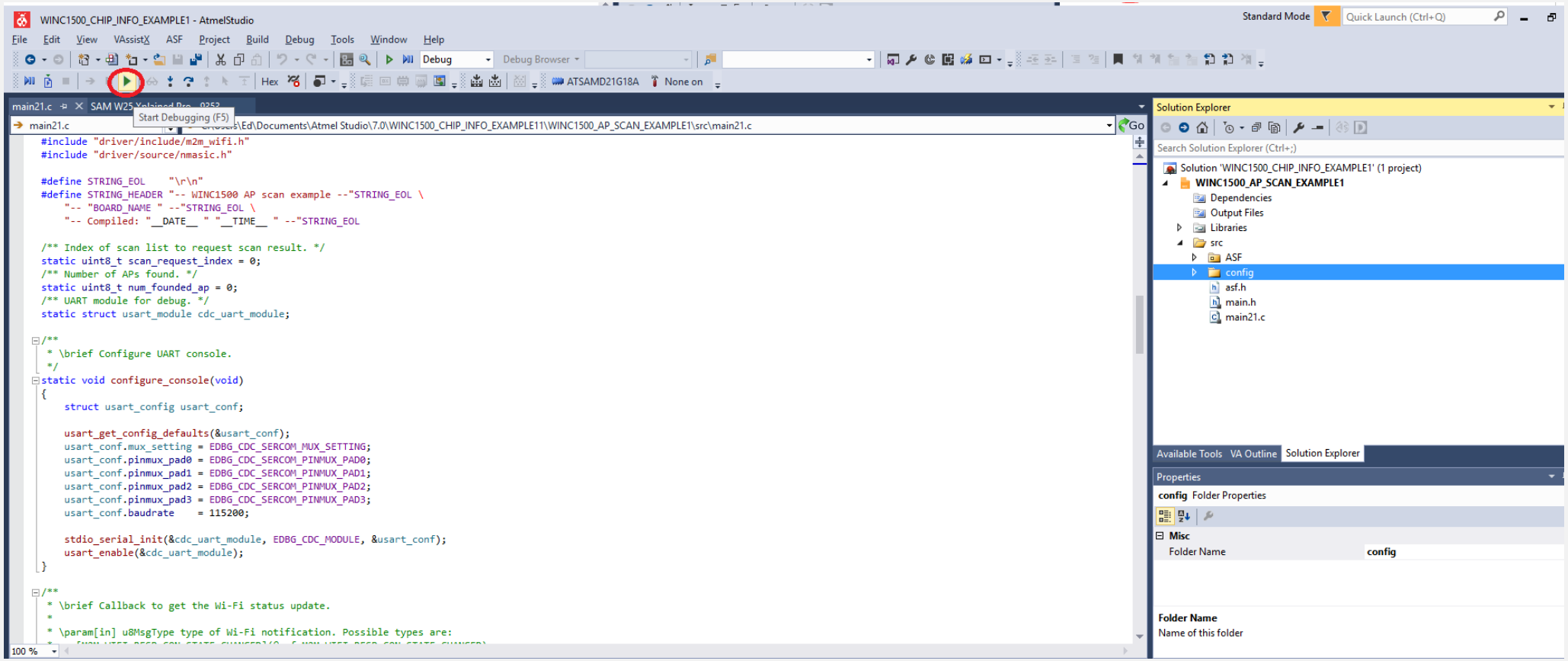


- Go back to the "SAMW25" examples and choose the "AP Scan Example"



RUNNING A WINC1500 EXAMPLE

- Let's run the code in debug mode and see what it does! Hit the "Debug" arrow on the top toolbar (or press F5). The IDE will compile and upload the code into the SAMD21



RUNNING A WINC1500 EXAMPLE – HOW TO SET A BREAKPOINT

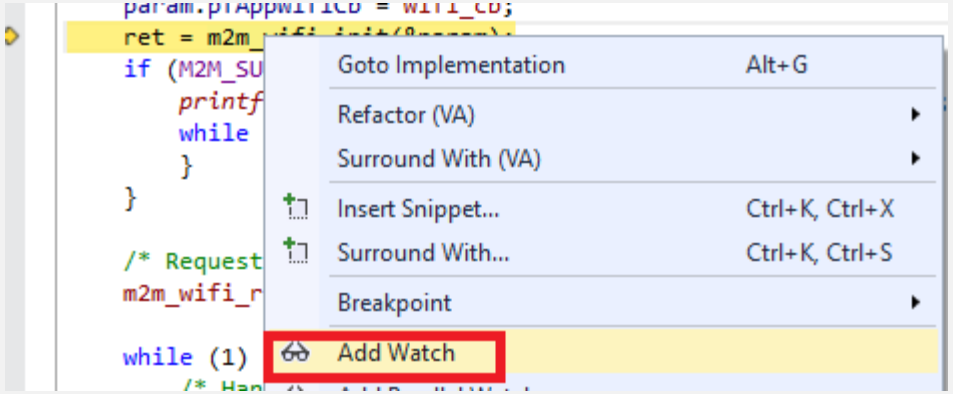
- The code will upload and instantly start running – You can tell by the status bar at the bottom that says "running"!

- Let's set a breakpoint and reinitialize the device. Open "main21.c" and on line 250 click on the left-column beside the code. A red dot, symbolizing the position of the breakpoint, will appear.

- On the debugging toolbar, hit "Reset" (Shift + f5) and then "Continue" (f5). The debugger must have hit the breakpoint. The yellow arrow indicates the current position of the MCU on the code.

RUNNING A WINC1500 EXAMPLE – HOW TO WATCH A VARIABLE

- To watch a variable, right click on a variable name and hit "Add Watch". This will add them to the watch window, where you can see their value change as you progress through the code.



- For example, add the variable "param" and put a breakpoint on line 250. Restart the code. Watch as the variable is assigned a value on lines 250-251

- You can also change the value of a variable – good for debugging (sometimes)

Watch 1

Name	Value
param	{...}
pfAppWifiCb	0x0 <exception_table>
pfAppMonCb	0x0 <exception_table>
strEthInitParam	

Line 250

Watch 1

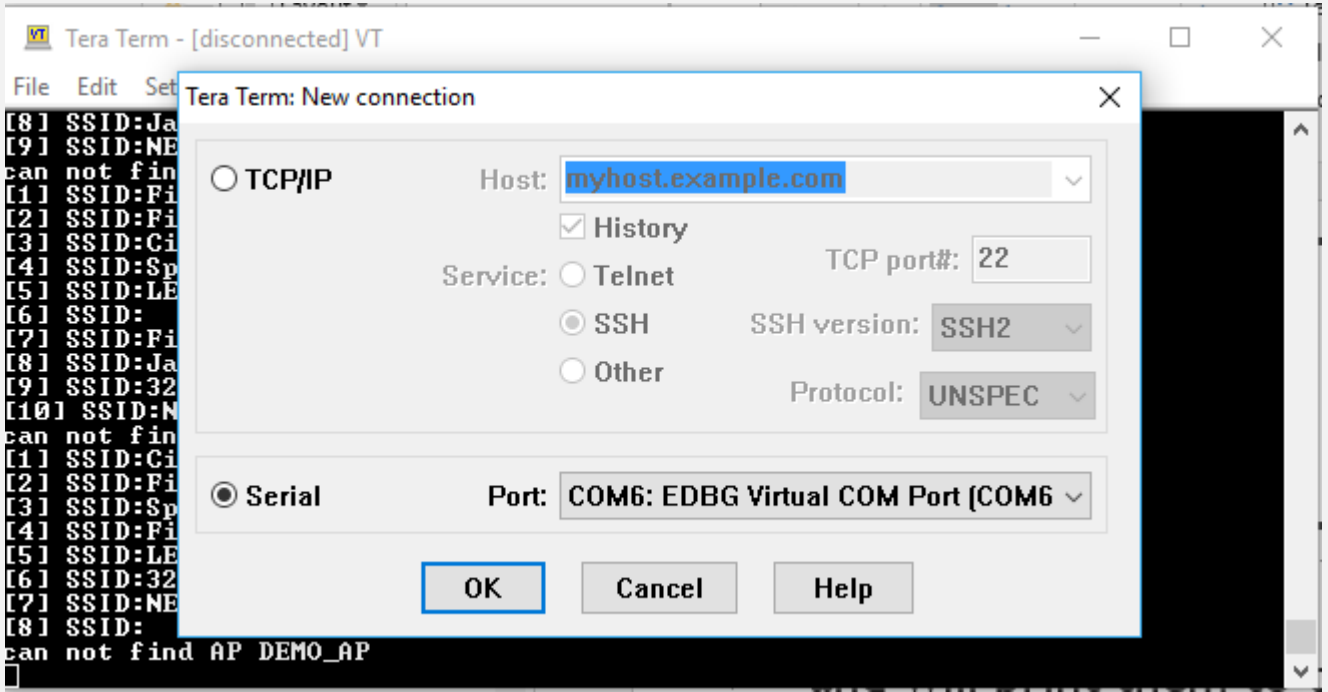
Name	Value
param	{...}
pfAppWifiCb	0x44c9 <wifi_cb>
pfAppMonCb	0x0 <exception_table>
strEthInitParam	

Line 252

CONTINUING WITH A WINC1500 EXAMPLE

- Let's continue with the example. As you can read in the comments, this example scans for APs (Internet Access Points) and will print them to the UART.

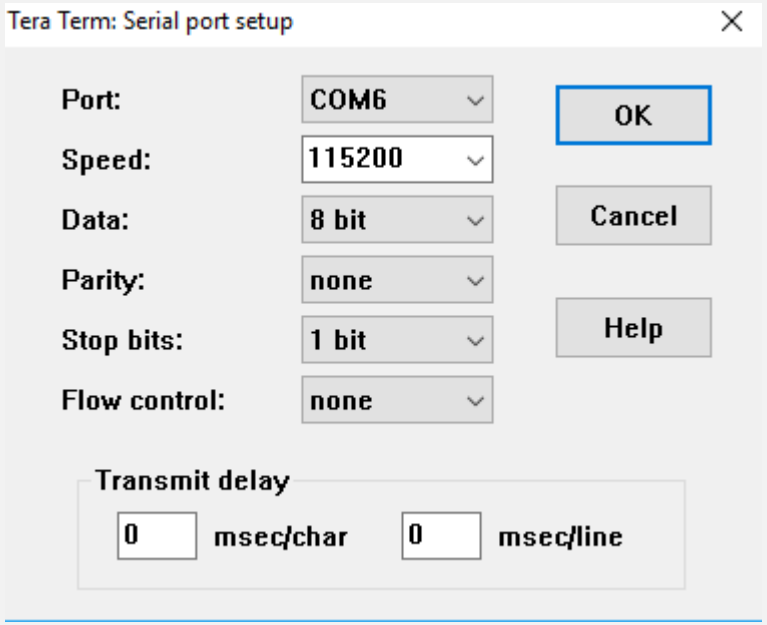
- To read the uart, open a terminal program (like TeraTerm) and set it up to listen the COM por of the SAMW25 board (in this case, it is COM6)

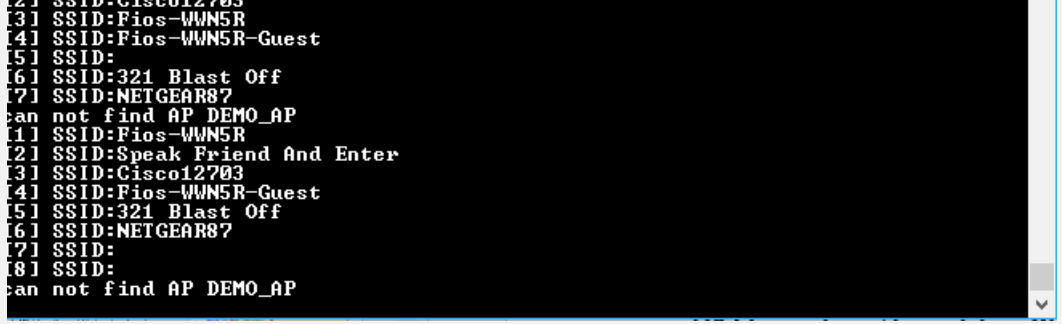


CONTINUING WITH A WINC1500 EXAMPLE

- Once TeraTerm is open, we need to set it up to the same BAUD RATE as the example uses – this is 115200 8N1
  - 8N1 is shorthand for "8 bit data, no stop bit, one parity bit – a old timey notation! More on 8N1 notation

  - Once set up, you should be able to see APs being displayed on the console!





MAKING YOUR OWN PROJECT FOR THE SAMW25 DEV BOARD

MAKING A BLANK PROJECT FOR THE SAMW25 DEV BOARD

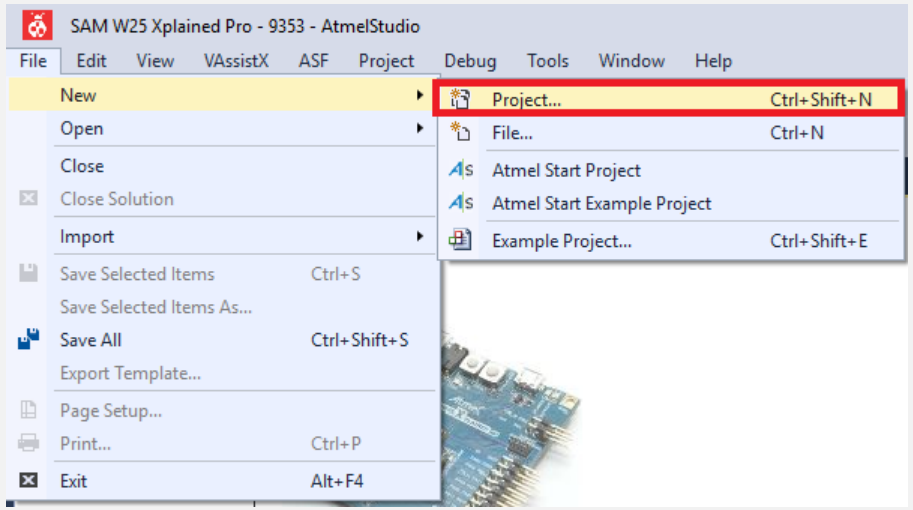
- You can also make you own project for the SAMW25 Dev board

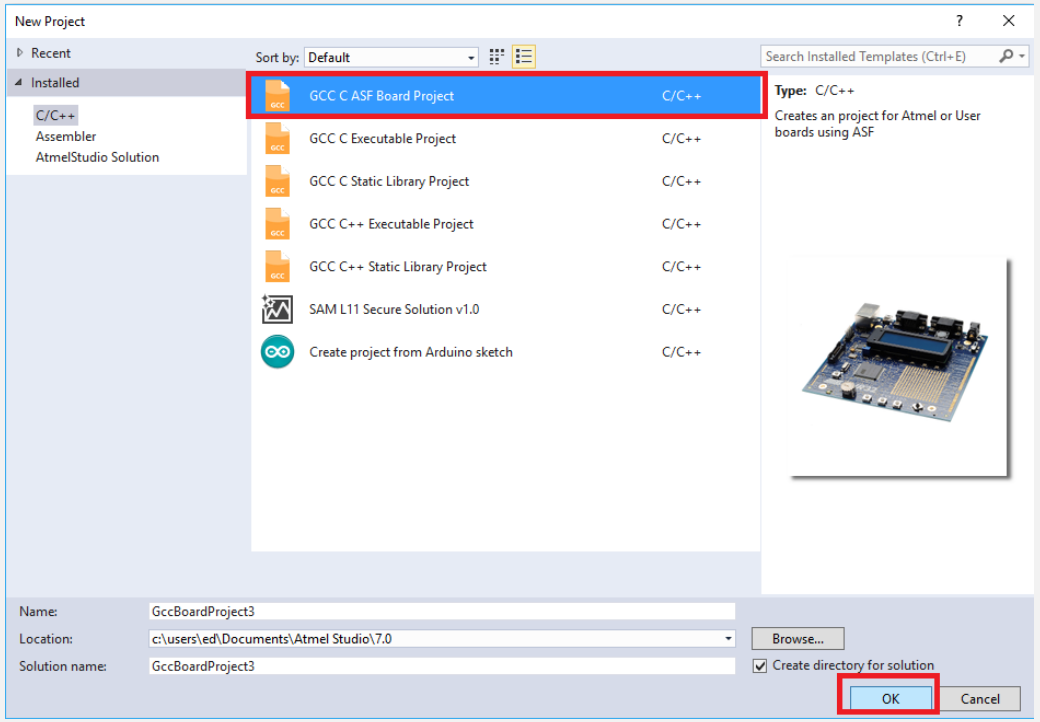
  - ***This takes into advantage the mapping of the physical board – It has a Board Support Package (BSP) that designates MCU pins to the hardware on the system (example – GPIO to LEDs on board, UART to on board EDBG bridge, etc).***

  - ***Later in the course we will see how to do a project for a completely custom board, such as the one you will be doing***

MAKING A BLANK PROJECT FOR THE SAMW25 DEV BOARD

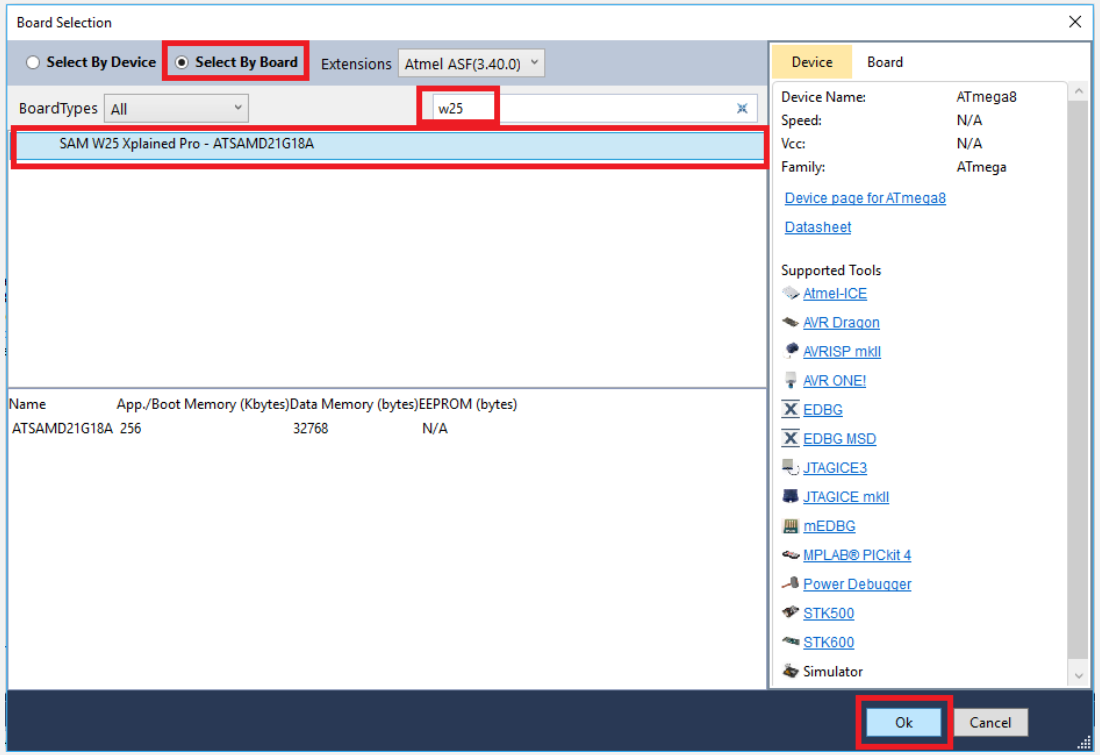
- Click on "New >> Project…"
- Choose "GCC C ASF Board Project, give it a name and location, and hit "Ok"

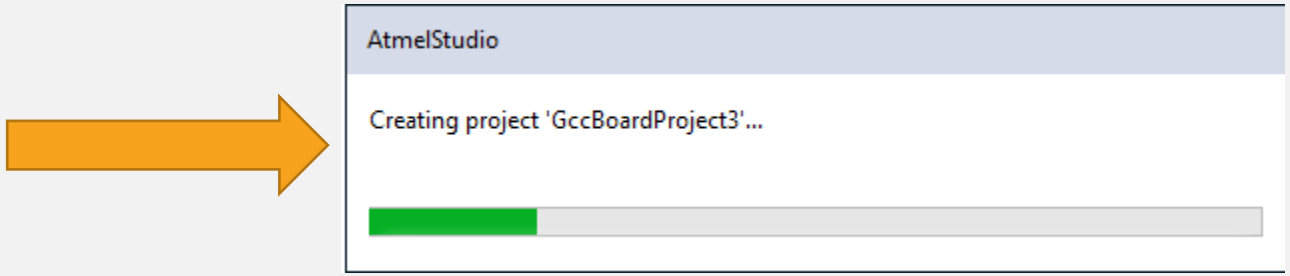




MAKING A BLANK PROJECT FOR THE SAMW25 DEV BOARD

- Choose "Select by Board" to show options based on Atmel Dev Boards
- Search for "w25"
- Select "Sam W25 Xplained Pro – ATSAMD21G18A" and hit "Ok". Atmel Studio will create a project



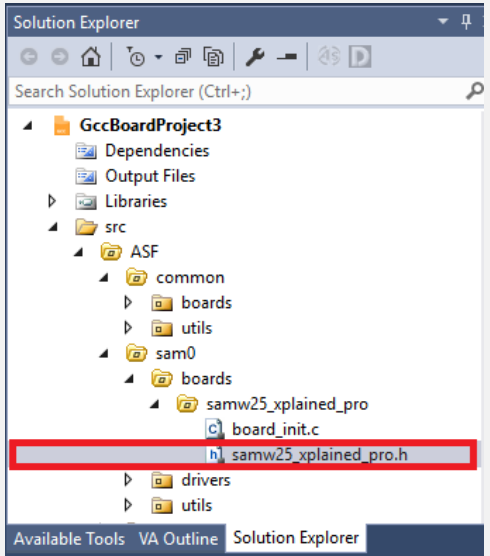


MAKING A BLANK PROJECT FOR THE SAMW25 DEV BOARD

- You will now have a blank project with very basic code to start with. Now, explore "main.c" and get familiar with what is done:

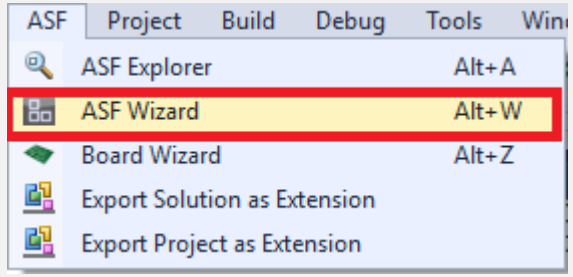
- - What does "system_init()" do? Where is it defined?
- What does the code in the while loop in main do?

- Go to "src >> ASF >> sam0 >> boards >> samw25_xplained_pro >> samw25_xplained_pro.h". What does this file do?

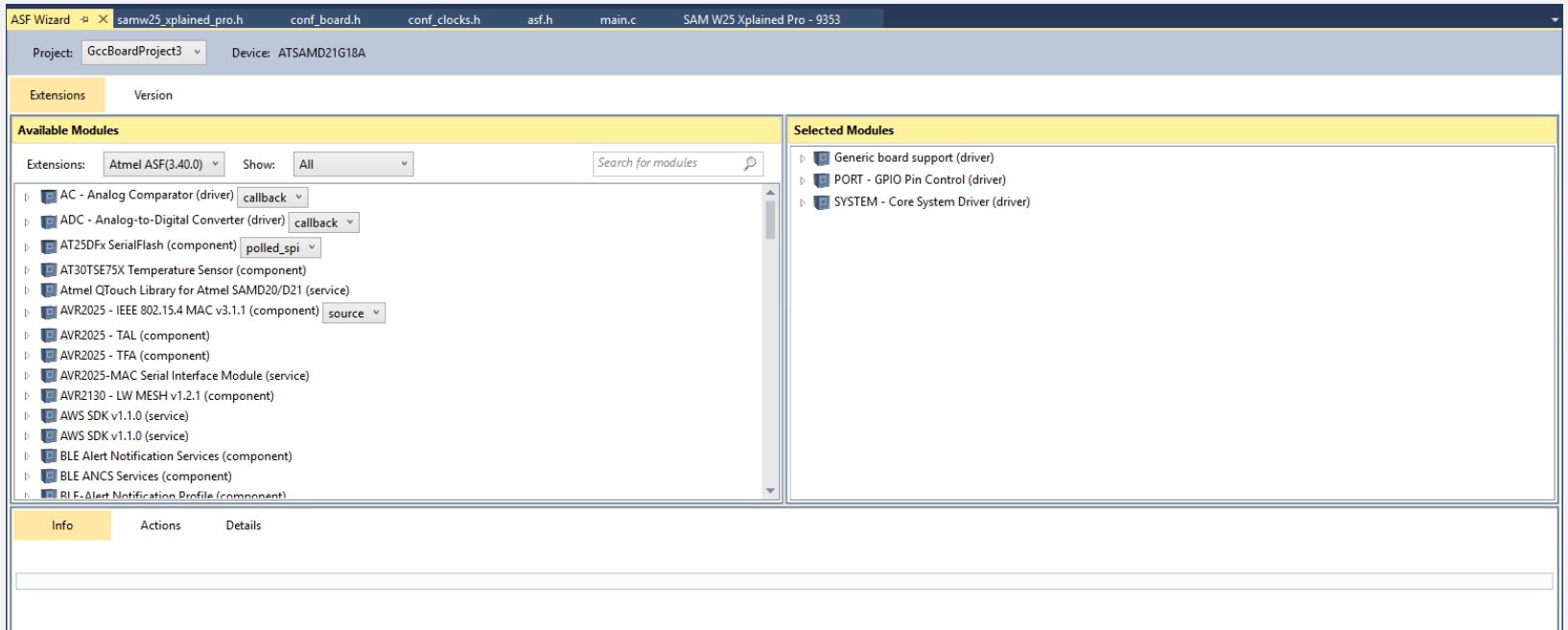


ADDING ASF MODULES TO YOUR SAMW25 DEV BOARD PROJECT

- Now that we have a blank project, let's add ASF modules to it!



- Go to "ASF >> ASF Wizard". This will open a window that will show all the ASF modules available for your platform (left) and the modules that are currently present in your project (right). Take some time to go through and read the modules ASF has for you.



FINDING EXAMPLES FOR ASF MODULES YOU JUST ADDED

- Microchip has multiple examples as well as API documentation online:
  - http://asf.atmel.com/docs/latest/samd21/html/index.html
- This will be one of the best resources for you in this class!
- To learn more about the UART driver we just added, read http://asf.atmel.com/docs/latest/samd21/html/group__asfdoc__sam0__sercom__usart__group.html#asfdoc_sam0_sercom_usart_special_considerations
- Implement the example mentioned on the following page in your project
- http://asf.atmel.com/docs/latest/samd21/html/asfdoc_sam0_sercom_usart_callback_use_case.html

FINDING EXAMPLES FOR ASF MODULES YOU JUST ADDED

- Read the previous example code and understand what it tries to do.
- Open a terminal program, like TeraTerm, and play with the example

- Comprehension questions
  - How many characters does it need before it replies a response?
  - Does this example allow for the device to perform other tasks in the meanwhile? Is it blocking?
  - How would you change it to make the system non-blocking?

CONCLUSION

- As you saw in the previous slide, finding toy examples and trying them out is a very common step in integrating new ASF modules into your code and making them work!

- The LAB0 assignment will be your fist programming challenge in this class and will be a warmup for the A1 assignment – the Command Line Interface

OFFICE HOURS

- We're finishing choosing these. Most likely a Wednesday + a weekend.
- These will also be on the class calendar.

- Keep using Piazza! Great community being developed already.